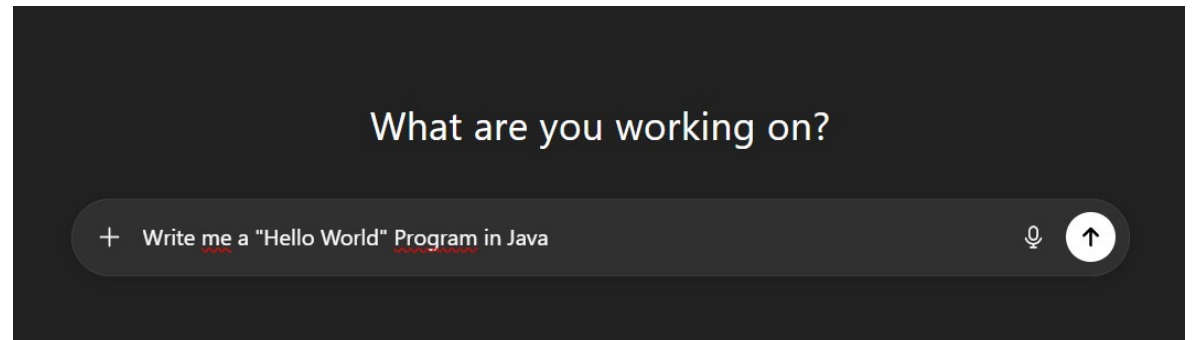




Früher



Heute

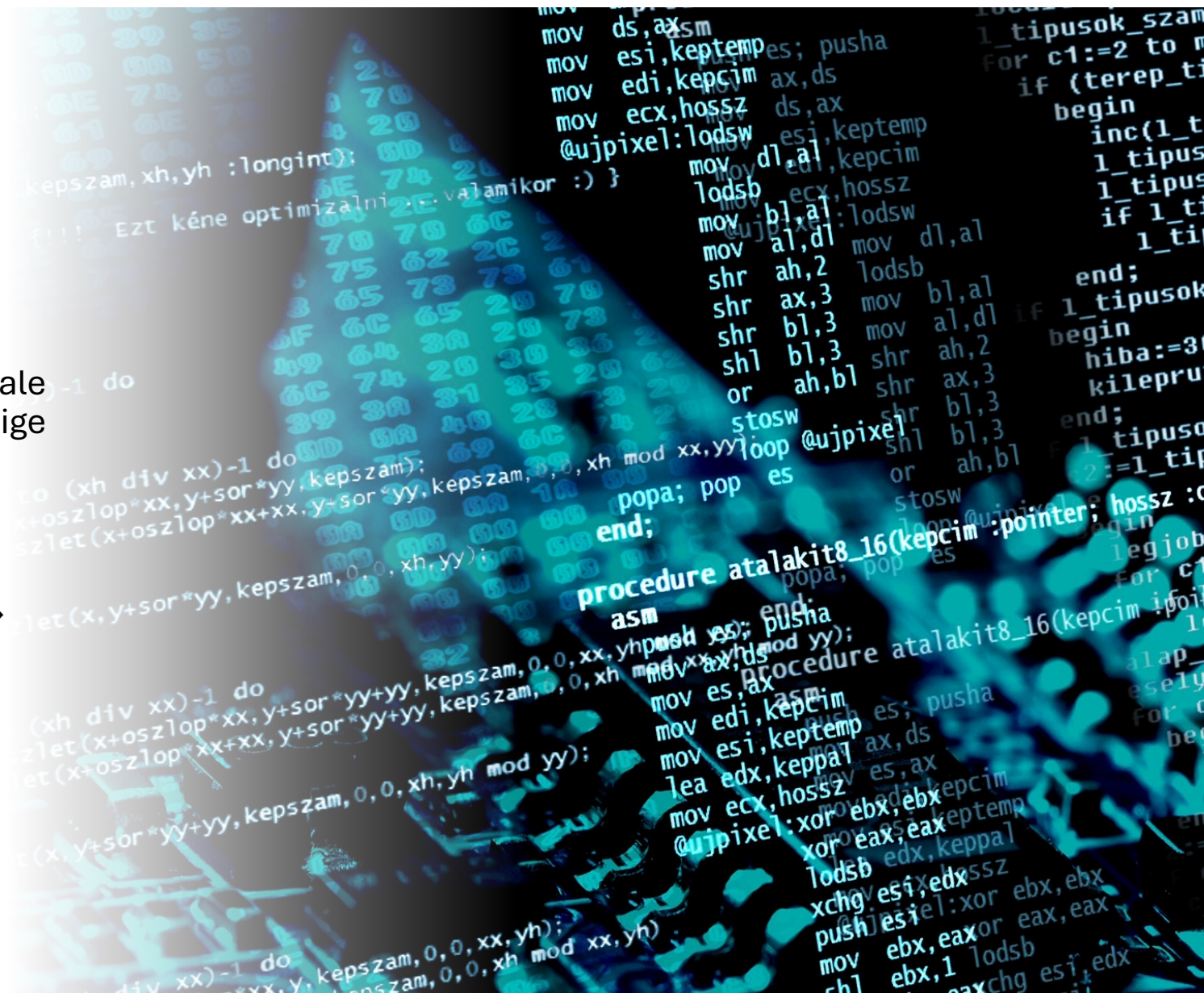


Zeitreise des Software-Engineerings

Team Bugisoft

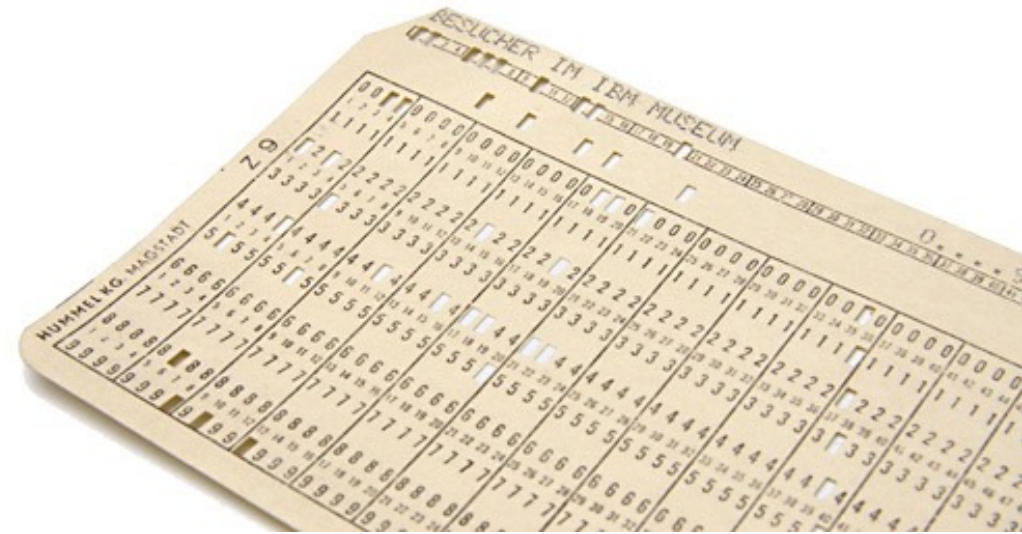
Einführung

- Ziel: Überblick über zentrale Meilensteine und zukünftige Entwicklungen
- Zeitraum: ca. 60 Jahre Softwaregeschichte
- Struktur: Vergangenheit → Zukunft



Meilenstein 1

- Erste elektronische Rechner: ENIAC (1945), Colossus
- Programmierung = direkte Verdrahtung und Steckerfelder
- Kein Konzept von „Software“ oder modularer Programmierung
- Programme waren extrem fehleranfällig und schwer reproduzierbar
- Grundlage für spätere Entwicklungen → erste Ansätze für systematisches Vorgehen



Meilenstein 2

Entstehung erster Programmiersprachen

- Ersten Compiler entstehen, Code muss nicht mehr in Maschinensprache geschrieben werden.
- Menschennähere Syntax statt Bit/Assembler
- weniger Hardwarebindung
- Frühe Standardbibliotheken und formale Beschreibungen
- Fortran (Wissenschaft), COBOL (Business), LISP (KI/funktional) → Paradigmenvielfalt, die bis heute nachwirkt.
- Basis für strukturiertes Programmieren, OOP und moderne Toolchains



Meilenstein 3

Strukturiertes Programmieren

- Kein Spaghetti-Code: klare Kontrollstrukturen Sequenz, Verzweigung, Schleife.
- Zerlegung in Funktionen/Prozeduren mit definierten Schnittstellen → Wiederverwendung & Teamarbeit
- bessere Lesbarkeit, Testbarkeit, Debugbarkeit; Top-Down-Entwurf, Schleifeninvarianten
- Nassi-Shneiderman-Diagramme, strukturierter Pseudocode; Prägung von Pascal und C
- Fundament für OOP, Clean-Code-Prinzipien und moderne Framework-/Architekturpraxis

Meilenstein 3

Strukturiertes Programmieren

```
!-----main program
use devObject
implicit none
!-----Executables
call open_devObjects
!...
call usersub
!...
call close_devObjects
end
!-----end of main

subroutine usersub
use devObject
implicit none

integer,parameter:: size=1024
type(devVar) dv1,dv2,dv3,dv4

real ar1d1(size),ar1d2(size),ar1d3(size), ar1dresult(size)

call random_number(ar1d1)
call random_number(ar1d2)
call random_number(ar1d3)

dv1=allocate_dv('real',size)
dv2=allocate_dv('real',size)
dv3=allocate_dv('real',size)

call transfer_r4(ar1d1,dv1,.true.)
call transfer_r4(ar1d2,dv2,.true.)
call transfer_r4(ar1d3,dv3,.true.)

!pointwise multiplication division addition subtraction test
dv4=(3.14159*dv1*(dv2+dv1)*(dv1-(.553+dv3))*dv2+(-.244))/dv1

call transfer_r4(ar1dresult,dv4,.false.)

call deallocate_dv(dv1)
call deallocate_dv(dv2)
call deallocate_dv(dv3)
call deallocate_dv(dv4)
end subroutine usersub
```

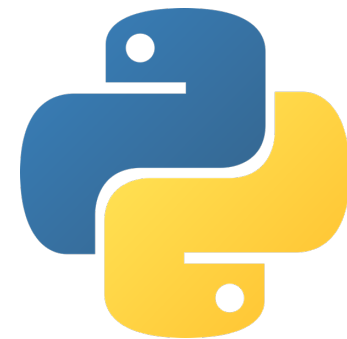
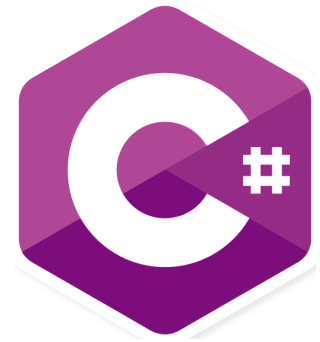


```
27 #include <iostream>
28 using namespace std;
29
30 class Animal {
31 public:
32     void speed() {
33         cout << "Who is more faster\n" ;
34     }
35 };
36
37 class Cheetah : public Animal {
38 public:
39     void speed() {
40         cout << "cheetah says im faster \n" ;
41     }
42 };
43
44 class Dolphin : public Animal {
45 public:
46     void speed() {
47         cout << "Dolphin says im faster \n" ;
48     }
49 };
50 int main() {
51     Animal a;
52     Cheetah c;
53     Dolphin d;
54
55     a.speed();
56     c.speed();
57     d.speed();
58     return 0;
59 }
```


Meilenstein 4

Objektorientiertes Programmieren

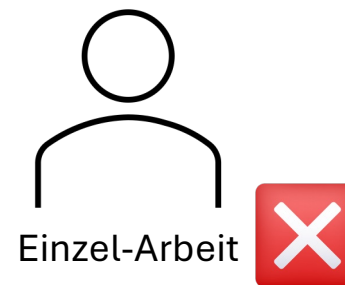
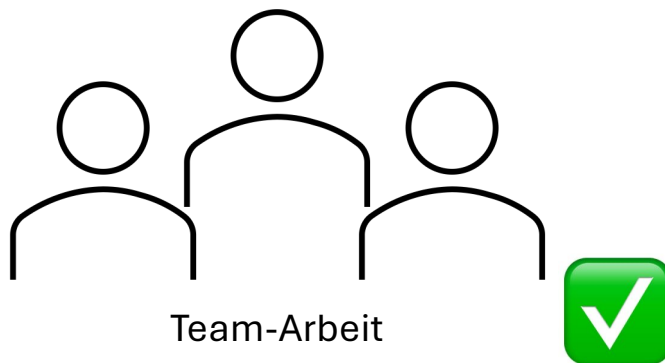
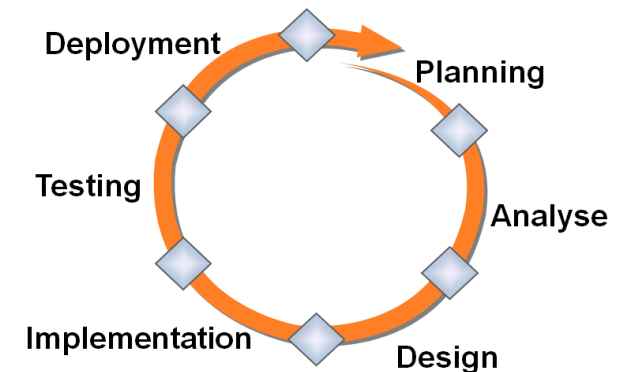
- Objekte bündeln Daten und Verhalten
- Modellierung orientiert sich an realer Welt
- Kapselung schützt interne Objektstruktur
- Vererbung fördert Wiederverwendung von Code
- Polymorphie ermöglicht flexible Schnittstellen
- Modularität erleichtert Wartung und Erweiterung
- Klare Strukturen verbessern Testbarkeit und Zusammenarbeit
- Basis für moderne Softwarearchitekturen
- Skalierbare Entwicklung statt Einzelprogrammierung



Meilenstein 5

Agile Methoden

- Fokus auf Flexibilität statt starrer Planung
- Zusammenarbeit wird zentraler Bestandteil der Entwicklung
- Iteratives Vorgehen ersetzt lange Projektzyklen
- Teams erhalten mehr Eigenverantwortung und Autonomie
- Anpassung an Veränderungen wird zur Stärke
- Methoden fördern schnelle und nachhaltige Ergebnisse



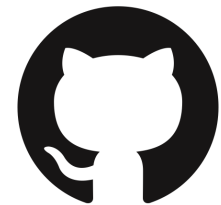
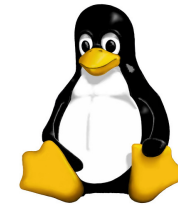
Meilenstein 6

Open Source & Cloud Computing

Open Source:

- Linux → Weitverbreitetes Betriebssystem
- Apache → dominierender Webserver
- GitHub → Social Coding, Kollaborationen

→ Wissen wurde demokratisiert und Innovationen beschleunigt



Cloud Computing:

- Start: Amazon Web Services (2006) → Infrastruktur „on demand“
 - Später: Google Cloud, Microsoft Azure
- Fördert Skalierbarkeit und Agilität



Zukunft 1

KI als Co-Programmierer

Heute schon sichtbar:

- GitHub Copilot → KI-gestützte Codevervollständigung
- ChatGPT & weitere LLMs → Generieren Tests, Dokumentationen etc.

Zukünftig:

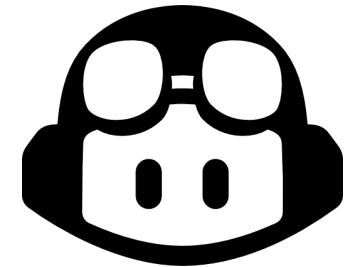
- KI-gestützte Architekturentscheidungen
- Debugging und Analysen automatisieren

Chancen:

- Mehr Produktivität
- Weniger Fehler, bessere Wertbarkeit

Risiken:

- Abhängigkeit von KI-Systemen
- Urheberrechtliche & ethische Fragen ©



Gib mir ein Beispiel Python Code

Klar! 😊

Hier ist ein einfaches Beispiel in **Python**, das die Summe aller geraden Zahlen von 1 bis 100 berechnet:

```
python
# Summe aller geraden Zahlen von 1 bis 100

# Initialisierung der Summe
summe = 0

# Schleife über alle Zahlen von 1 bis 100
for zahl in range(1, 101):
    if zahl % 2 == 0: # Prüfen, ob die Zahl gerade ist
        summe += zahl

# Ergebnis ausgeben
print("Die Summe aller geraden Zahlen von 1 bis 100 ist:", summe)
```

Code kopieren

💡 **Erklärung:**

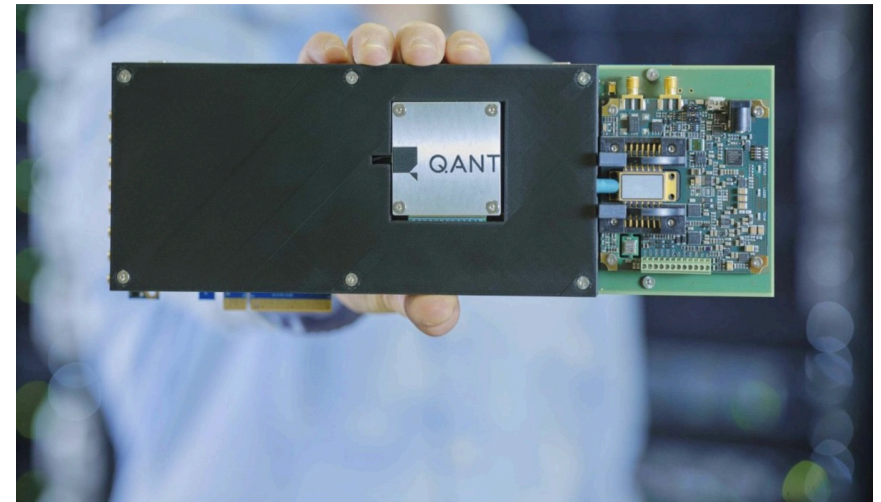
- `range(1, 101)` erzeugt alle Zahlen von 1 bis 100 (101 ist exklusiv).
- `zahl % 2 == 0` prüft, ob eine Zahl gerade ist.
- Das Ergebnis wird aufsummiert und am Ende ausgegeben.

Möchtest du, dass ich dir ein Beispiel mit einer bestimmten **Thematik** zeige (z. B. Dateioperation, Webanfrage, GUI, Mathe, KI usw.)?

Zukunft 2

Lichtbasierte Prozessoren

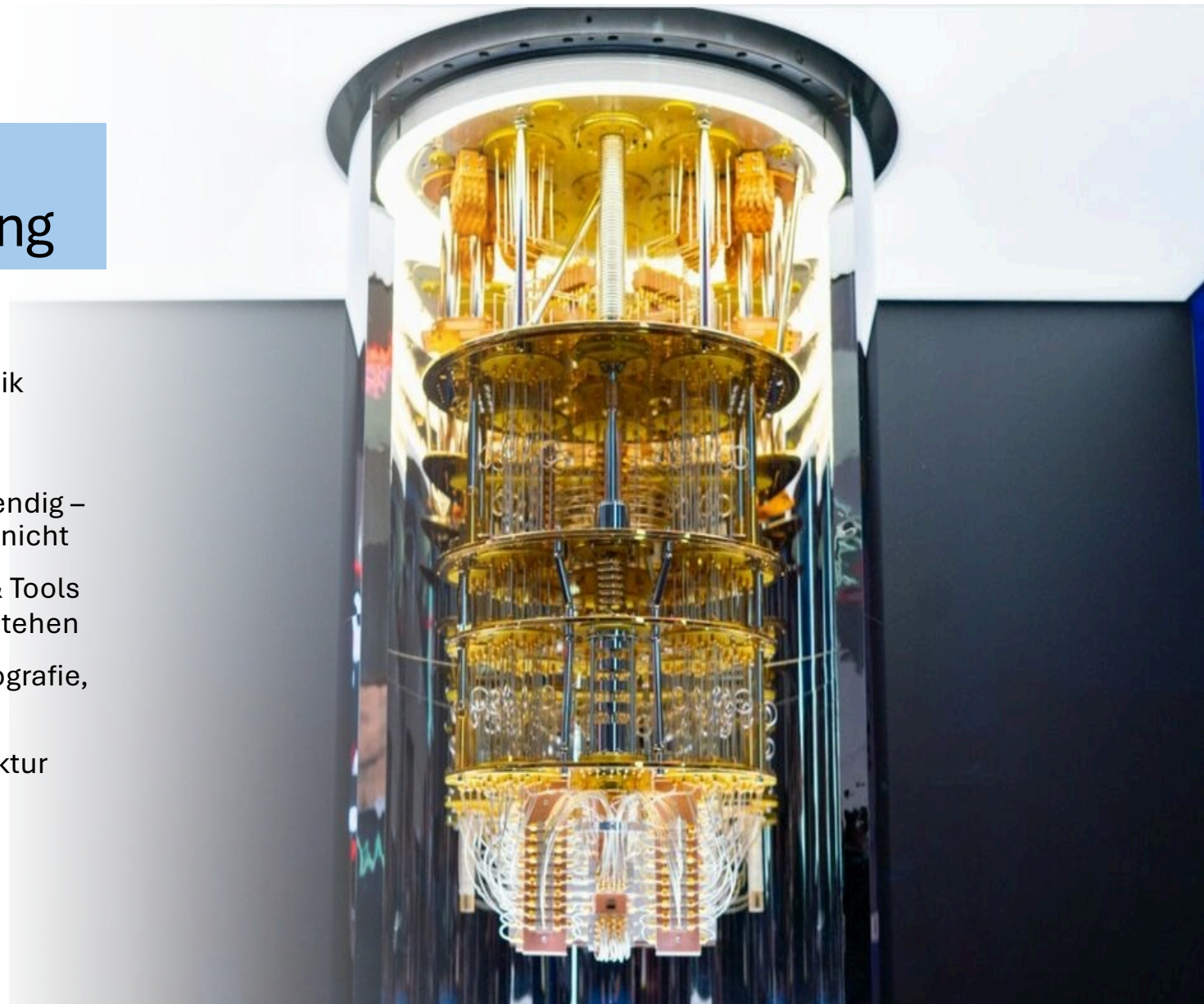
- Datenverarbeitung mit Photonen statt Elektronen
→ kaum Wärme, extreme Geschwindigkeit
- Ermöglicht neue Architekturen jenseits der klassischen Transistor-Skalierung (Moore's Law)
- Software muss massiv parallelisiert und für neuartige Hardware optimiert werden
- Potenzial: Echtzeit-Analyse von Big Data, KI-Training in Sekunden statt Wochen
- Herausforderung: Übergang von elektronischer zur hybriden photonischen Softwareumgebung



Zukunft 3

Quantencomputing

- Rechner, die Quantenmechanik nutzen (Superposition, Verschränkung)
- Völlig neue Algorithmen notwendig – klassische Software läuft hier nicht
- Neue Programmiersprachen & Tools (z.B. Q#, Cirq, IBM Qiskit) entstehen
- Potenzial: Revolution in Kryptografie, Optimierung, Simulationen
- Herausforderung: Fehlerkorrektur und völlig andere Debugging-Konzepte



Zusammenfassung

Vergangenheit:

1. 1940er: ENIAC, Colossus – keine echte Software, Verdrahtung
2. 1950er: Erste Sprachen – Fortran, COBOL, LISP
3. 1970er: Strukturiertes Programmieren → klare Logik & Funktionen
4. 1980er: Objektorientierung (OOP) → Objekte, Wiederverwendung
5. 2000er: Agile Methoden → flexibel, teamorientiert
6. 2010er: Open Source & Cloud → Skalierbarkeit, Kollaboration

Zukunft:

- **KI-Co-Programmierer** – Automatisierung, höhere Produktivität
- **Photonen-Prozessoren** – extrem schnell, neue Softwaremodelle
- **Quantencomputing** – neue Algorithmen, revolutionäre Leistung

Fazit

- Software-Engineering von Hardware-naher Einzelarbeit
→ hochgradig kollaborativen, KI-gestützten & cloudbasierten Ökosystem
- Zukünftig: KI, Photonik & Quantencomputing die Branche grundlegend verändern →
 - schaffen neue Paradigmen
 - Programmierung & Hardware „verschmelzen“